(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 December 2003 (24.12.2003)

PCT

(10) International Publication Number
WO 03/107576 A2

(51) International Patent Classification[7]: H04L

(21) International Application Number: PCT/US03/18788

(22) International Filing Date: 13 June 2003 (13.06.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/389,052 13 June 2002 (13.06.2002) US

(71) Applicant: **CERISENT CORPORATION** [US/US]; 2000 Alameda de las Pulgas, Suite 100, San Mateo, CA 94403-1269 (US).

(72) Inventors: **LINDBLAD, Christopher**; 35 Live Oak Road, Berkeley, CA 94705 (US). **PEDERSEN, Paul**; 1788 Oak Creek #310, Palo Alto, CA 94304 (US).

(74) Agents: **ALBERT, Philip, H.** et al.; Townsend and Townsend and Crew LLP, Two Embarcadero Center, 8th Floor, San Francisco, CA 94111 (US).

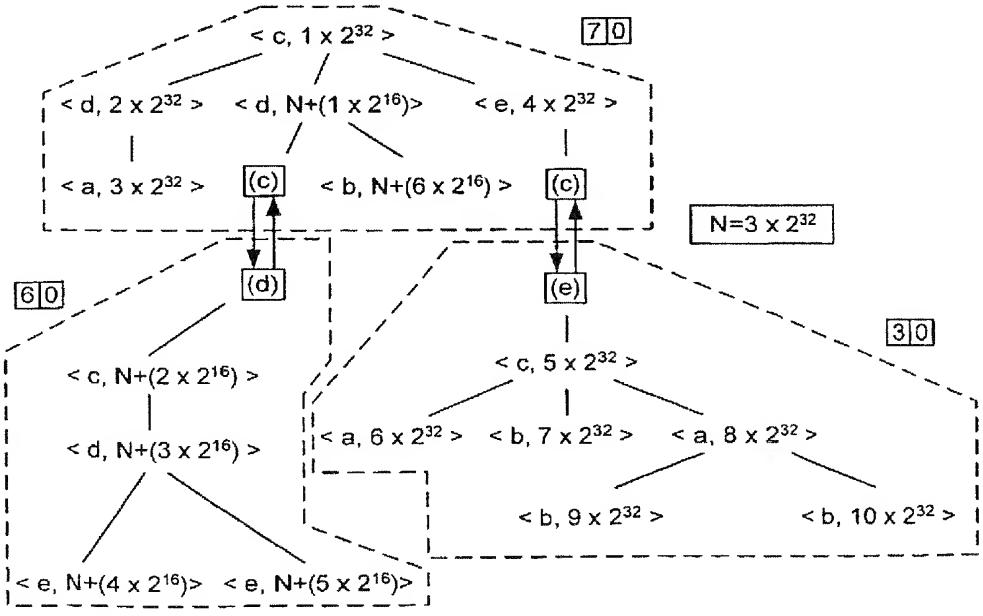
(81) Designated States *(national)*: AE, AG, AL, AM, AT (utility model), AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ (utility model), CZ, DE (utility model), DE, DK (utility model), DK, DM, DZ, EC, EE (utility model), EE, ES, FI (utility model), FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK (utility model), SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**
*without international search report and to be republished upon receipt of that report*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

(54) Title: XML-DB TRANSACTIONAL UPDATE SYSTEM




(57) Abstract: In an XML handling system, point updates to an element of an XML document stored in the database is possible. Updates include addition or deletion of whole documents, addition of a child node to any element node (this includes attribute nodes), the addition of new siblings to any element node, the deletion of any element node, and the replacement of any node by a new node. The database system might include a set of functions that can be invoked to affect an update (i.e., an addition, deletion or modification). Such updates can be submitted as queries, such as instructions within an XQuery query.

WO 03/107576 A2

# XML-DB TRANSACTIONAL UPDATE SYSTEM

## CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 60/389,052, filed June 13, 2002, entitled "XML DB TRANSACTIONAL UPDATE SYSTEM," which disclosure is incorporated herein by reference for all purposes.

The present disclosure is related to the following commonly assigned co pending U.S. Patent Applications:

No. _____ (Attorney Docket No. 021512 000110US, filed on the same date as the present application, entitled "A SUBTREE STRUCTURED XML DATABASE" (hereinafter "Lindblad I-A");

No. _____ (Attorney Docket No. 021512 000210US, filed on the same date as the present application, entitled "PARENT-CHILD QUERY INDEXING FOR XML DATABASES" (hereinafter "Lindblad II-A"); and

No. _____ (Attorney Docket No. 021512 000410US, filed on the same date as the present application, entitled "XML DATABASE MIXED STRUCTURAL-TEXTUAL CLASSIFICATION SYSTEM" (hereinafter "Lindblad IV-A");

The respective disclosures of these applications are incorporated herein by reference for all purposes.

## BACKGROUND OF THE INVENTION

### Field of the Invention

[0002] This invention relates in general to updating structured databases such as XML databases on a network, and more specifically, to updating one or more subtree-structured XML databases over a network.

### Description of Related Art

[0003] Extensible Markup Language (XML) is a restricted form of SGML, the Standard Generalized Markup Language defined in ISO 8879 and XML is one form of structuring data. XML is more fully described in "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation (6 October 2000), which is incorporated by reference herein for all purposes [and available at http://www.w3.org/TR/2000/REC-xml-20001006] (hereinafter, "XML Recommendation"). XML is a useful form of structuring data because it is an open format that is human-readable

and machine-interpretable. Other structured languages without these features or with similar features might be used instead of XML, but XML is currently a popular structured language used to encapsulate (obtain, store, process, etc.) data in a structured manner.

**[0004]** An XML document has two parts: 1) a markup document and 2) a document schema. The markup document and the schema are made up of storage units called "elements", which can be nested to form a hierarchical structure. An example of an XML markup document 10 is shown in Fig. 1. Document 10 (at least the portions shown) contains data for one "citation" element. The "citation" element has within it a "title" element, and "author" element and an "abstract" element. In turn, the "author" element has within it a "last" element (last name of the author) and a "first" element (first name of the author). Thus, an XML document comprises text organized in freely-structured outline form with tags indicating the beginning and end of each outline element.

**[0005]** Generally, an XML document comprises text organized in freely-structured outline form with tags indicating the beginning and end of each outline element. In XML, a tag is delimited with angle brackets followed by the tag's name, with the opening and closing tags distinguished by having the closing tag beginning with a forward slash after the initial angle bracket.

**[0006]** Elements can contain either parsed or unparsed data. Only parsed data is shown for document 10. Unparsed data is made up of arbitrary character sequences. Parsed data is made up of characters, some of which form character data and some of which form markup. The markup encodes a description of the document's storage layout and logical structure. XML elements can have associated attributes, in the form of name-value pairs, such as the publication date attribute of the "citation" element. The name-value pairs appear within the angle brackets of an XML tag, following the tag name.

**[0007]** XML schemas specify constraints on the structures and types of elements and attribute values in an XML document. The basic schema for XML is the XML Schema, which is described in "XML Schema Part 1: Structures", W3C Working Draft (24 September 1999), which is incorporated by reference herein for all purposes [and available at http://www.w3.org/TR/1999/WD-xmlschema-1-19990924]. A previous and very widely used schema format is the DTD (Document Type Definition), which is described in the XML Recommendation.

**[0008]** Since XML documents are typically in text format, they can be searched using conventional text search tools. However such tools might ignore the information content provided by the structure of the document, one of the key benefits of XML. Several query

languages have been proposed for searching and reformatting XML documents that do consider the XML documents as structured documents. One such language is XQuery, which is described in "XQuery 1.0: An XML Query Language", W3C Working Draft (20 December 2001), which is incorporated by reference herein for all purposes [and available at http://www.w3.org/TR/XQuery]. An example of a general form for an XQuery query is shown in Fig. 2. Note that the ellipses at line [03] indicate the possible presence of any number of additional namespace prefix to URI mappings, the ellipses at line [12] indicate the possible presence of any number of additional function definitions and the ellipses at line [17] indicate the possible presence of any number of additional FOR or LET clauses.

**[0009]** XQuery is derived from an XML query language called Quilt [described at http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html], which in turn borrowed features from several other languages, including XPath 1.0 [described at http://www.w3.org/TR/XPath.html], XQL [described at Http://www.w3.org/TandS/QL/QL98/pp/xql.html], XML-QL [described at http://www.research.att.com/~mff/files/final.html] and OQL.

**[0010]** Query languages predated the development of XML and many relational databases use a standardized query language called SQL, as described in ISO/IEC 9075-1:1999. The SQL language has established itself as the *linqua franca* for relational database management and provides the basis for systems interoperability, application portability, client/server operation, and distributed databases. XQuery is proposed to fulfill a similar same role with respect to XML database systems. As XML becomes the standard for information exchange between peer data stores, and between client visualization tools and data servers, XQuery may become the standard method for storing and retrieving data from XML databases.

**[0011]** With SQL query systems, much work has been done on the issue of efficiency, such as how to process a query, retrieve matching data and present that to the human or computer query issuer with efficient use of computing resources to allow responses to be quickly made to queries. As XQuery and other tools are relied on more and more for querying XML documents, efficiency will be more essential.

## BRIEF SUMMARY OF THE INVENTION

**[0012]** An update system as described herein applies updates to XML nodes in an XML database. In an XML handling system according to one embodiment of the present invention, point updates to an element of an XML document stored in the XML database are

possible. Updates might add or delete whole documents, add child nodes to a parent node where the child node is another XML element or an attribute of an existing XML element, adding new siblings to a node, deleting a node, replacement of a node by a new node, etc.

**[0013]** In a particular implementation, a database system includes a set of functions that can be invoked to affect an update (i.e., an addition, deletion or modification). Such updates can be submitted as queries, such as instructions within an XQuery query.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0014]** Fig. 1 is an illustration of XML markup.

**[0015]** Fig. 2 is an illustration of an XQuery query.

**[0016]** Fig. 3 is an illustration of a simple XML document including text and markup.

**[0017]** Fig. 4 is a schematic representation of the XML document shown in Fig. 3; Fig. 4A illustrates a complete representation the XML document and Fig. 4B illustrates a subtree of the XML document.

**[0018]** Fig. 5 is a schematic representation of a more concise XML document.

**[0019]** Fig. 6 illustrates a portion of an XML document that includes tags with attributes; Fig. 6A shows the portion in XML format; Fig. 6B is a schematic representation of that portion in graphical form.

**[0020]** Fig. 7 shows a more complex example of an XML document, having attributes and varying levels.

**[0021]** Fig. 8 is a schematic representation of the XML document shown in Fig. 7, omitting data nodes.

**[0022]** Fig. 9 illustrates one decomposition of the XML document illustrated in Figs. 7-8.

**[0023]** Fig. 10 illustrates the decomposition of Fig. 9 with the addition of link nodes.

**[0024]** Fig. 11 is a block diagram of an XML handling system according to aspects of the present invention.

**[0025]** Fig. 12 is a flowchart of a process for performing an update to an XML database.

**[0026]** Fig. 13 illustrates a situation where an update encounters a link node.

**[0027]** Fig. 14 illustrates a method for updating subtree node counts and subtree ID slack.

**[0028]** Fig. 15 illustrates a subtree that represents a new document fragment.

**[0029]** Fig. 16 illustrates a subtree corresponding to the new document fragment of Fig. 15.

**[0030]** Fig. 17 illustrates new subtrees corresponding to the new document fragment of Fig. 15 and pre-existing nodes.

**[0031]** Fig. 18 represents a document fragment; Fig. 18A shows it in tree form; Fig. 18B shows it in XML form.

**[0032]** Fig. 19 illustrates a process for assigning ordinal values to nodes before and during an update step.

## DETAILED DESCRIPTION OF THE INVENTION

**[0033]** This detailed description illustrates some embodiments of the invention and variations thereof, but should not be taken as a limitation on the scope of the invention. In this description, structured documents are described, along with their processing, storage and use, with XML being the primary example. However, it should be understood that the invention might find applicability in systems other than XML systems, whether they are later-developed evolutions of XML or entirely different approaches to structuring data.

### Overview

**[0034]** Systems for generating and managing XML databases are described in Lindblad I-A. The nodes may be of any type, such as element nodes, attribute nodes, text nodes, processing instruction nodes or comment nodes. The notation *u(n)* is used herein to indicate an update operation *u* applied to the node *n*. "Elements" are generally understood in the context of XML documents, but would also apply where the data being manipulated is other than XML documents. As used herein, an XML element comprises a tag name, zero or more attribute (name, value) pairs, and element content. Element content is typically zero or more characters of text and zero or more child elements, but element content might take other forms.

**[0035]** An update system as described herein applies updates to XML nodes in an XML database. In an XML handling system according to one embodiment of the present invention, point updates to an element of an XML document stored in the XML database are possible. Updates might add or delete whole documents, add child nodes to a parent node where the child node is another XML element or an attribute of an existing XML element, adding new siblings to a node, deleting a node, replacement of a node by a new node, etc.

**[0036]** Lindblad I-A describes how a collection of XML documents might be decomposed into a "forest" of "subtrees", where each subtree describes a fragment within one of the XML documents.

Subtrees, Storage and Decomposition

**[0037]** Subtree storage is described in this section. Subtree storage is explained with reference to a simple example, but it should be understood that such techniques are equally applicable to more complex examples.

**[0038]** Fig. 3 illustrates an XML document 30, including text and markup. Fig. 4A illustrates a schematic representation 32 of XML document 30, wherein schematic representation 12 is a shown as a tree (a connected acyclic simple directed graph) with each node of the tree representing an element of the XML document or an element's content, attribute, the value, etc.

**[0039]** In a convention used for the figures of the present application, directed edges are oriented from an initial node that is higher on the page than the edge's terminal node, unless otherwise indicated. Nodes are represented by their labels, often with their delimiters. Thus, the root node in Fig. 4A is a "citation" node represented by the label delimited with "<>". Data nodes are represented by rectangles. In many cases, the data node will be a text string, but other data node types are possible. In many XML files, it is possible to have a tag with no data (e.g., where a sequence such as "<tag></tag>" exists in the XML file). In such cases, the XML file can be represented as shown in Fig. 4A but with some nodes representing tags being leaf nodes in the tree. The present invention is not limited by such variations, so to focus explanations, the examples here assume that each "tag" node is a parent node to a data node (illustrated by a rectangle) and a tag that does not surround any data is illustrated as a tag node with an out edge leading to an empty rectangle. Alternatively, the trees could just have leaf nodes that are tag nodes, for tags that do not have any data.

**[0040]** As used herein, "subtree" refers to a set of nodes with a property that one of the nodes is a root node and all of the other nodes of the set can be reached by following edges in the orientation direction from the root node through zero or more non-root nodes to reach that other node. A subtree might contain one or more overlapping nodes that are also members of other "inner" or "lower" subtrees; nodes beyond a subtree's overlapping nodes are not generally considered to be part of that subtree. The tree of Fig. 4A could be a subtree, but the subtree of Fig. 4B is more illustrative in that it is a proper subset of the tree illustrated in Fig. 4A.

**[0041]** To simplify the following description and figures, single letter labels will be used, as in Fig. 5. Note that even with the shorted tags, tree 35 in Fig. 5 represents a document that has essentially the same structure as the document represented by the tree of Fig. 4A.

**[0042]** Some nodes may contain one or more attributes, which can be expressed as (name, value) pairs associated with nodes. In graph theory terms, the directed edges come in two flavors, one for a parent-child relationship between two tags or between a tag and its data node, and one for linking a tag with an attribute node representing an attribute of that tag. The latter is referred to herein as an "attribute edge". Thus, adding an attribute (key, value) pair to an XML file would map to adding an attribute edge and an attribute node, followed by an attribute value node to a tree representing that XML file. A tag node can have more than one attribute edge (or zero attribute edges). Attribute nodes have exactly one descendant node, a value node, which is a leaf node and a data node, the value of which is the value from the attribute pair.

**[0043]** In the tree diagrams used herein, attribute edges sometimes are distinguished from other edges in that the attribute name is indicated with a preceding "@". Fig. 6A illustrates a portion of XML markup wherein a tag T has an attribute name of "K" and a value of "V". Fig. 6B illustrates a portion of a tree that is used to represent the XML markup shown in Fig. 6A, including an attribute edge 36, an attribute node 37 and a value node 38. In some instances, tag nodes and attribute nodes are treated the same, such as indexing sequences and the like, but other times are treated differently. To easily distinguish tag nodes and attribute nodes in the illustrated trees, tag nodes are delimited with surrounding angle brackets ("<>"), while attribute nodes are be limited with an initial "@".

Updating

**[0044]** Using such a structure for storing XML documents allows for dynamically updating an XML database of XML subtrees. Updates might include XML node deletion, replacement, and insertion. Nodes can be inserted as preceding siblings, following siblings, or as a new children nodes. Document nodes may be inserted or deleted. Nodes may be element nodes, attribute nodes, text nodes, processing instruction nodes or comment nodes.

**[0045]** Fig. 11 is a block diagram of an XML handling system 100 that is amenable to updating XML databases. As illustrated there, XML handling system 100 might accept XML documents 112 using a data loader 114 data populates an XML subtree database 116 with subtrees representing portions of the accepted XML documents. XML handling system 100

might also accept update requests 118 via an updater 120. XML handling system 100 might also accept queries via a query processor 122.

**[0046]** In a particular implementation, a database system includes a set of functions that can be invoked to affect an update (i.e., an addition, deletion or modification). Such updates can be submitted as queries, such as instructions within an XQuery query, in which case query processor 122 would absorb the role of updater 120.

**[0047]** One basic update-related operation of XML handling system 100 is to locate the subtree $S(n)$ containing the node $n$ that is the target of an update and then to create an updated copy $S'(n)$ of $S(n)$. Once fully created, with all unaffected nodes copied, deleted nodes removed, and new nodes added, the obsolete subtree $S(n)$ is atomically (transactionally) replaced in the database with the new, updated copy $S'(n)$. The process of replacing $S(n)$ with $S'(n)$ may require a cascading sequence of additional updates as described below.

**[0048]** XML handling system 100 might use a two-step process for updates. First, updates are specified using a set of update value constructors. Secondly, the update values are committed to the database by a commit function. Each of these is described in more detail below.

**[0049]** The commit function accepts as input any sequence of values, including some number of update values; it performs the updates as a side-effect, and returns the non-update values as a result sequence. Any XQuery expression that includes some calls to the update value constructors and returns a sequence of values including some number of update values will be automatically passed to the commit function.

**[0050]** The update values appearing in the input to commit are processed concurrently and transactionally, which means that no assumptions may or need to be made about the order in which the updates are performed, and that either all the updates complete consistently, or none of them completes and the database remains in the state it was in prior to the call to commit. XML handling system 100 detects deadlock conditions that may occur as a consequence of committing competing sets of update values.

**[0051]** One such process for performing an update is illustrated in Fig. 12. As shown there, the system inputs an update value constructor (S1) and accumulates update values (S2). Accumulating input values involves scanning a commit input sequence and extracting update values therefrom. Next, the target nodes are sorted into document order (S3), resulting in a change vector containing the update values sorted by document order for the target nodes of the updates. DocNode updates (i.e., complete XML document updates) are disposed (S4) of

because they do not involve interacting with a subtree structure - that just involves replacing a document.

**[0052]** Inconsistencies in the sets of update values appearing in the change vector are then detected and corrected (S5). Examples include *node-replace* requests that would update a descendant of replaced node and *node-insert-child* requests that conflict with a delete or replace of any ancestor of the node that is the target of the *node-insert-child* request.

**[0053]** Next, the node updates are processed (S6) and results reported (S7). The node update process loops over the ordered, non-conflicting node updates and creates all new updated subtrees. This step might use one or more of the subroutines shown in Appendix A.

Update Methods

**[0054]** The following section describes some methods that a database update module might employ in connection with updates. While the methods here are described with reference to XML documents, which are generally text files, the database system might accommodate other forms of structured data. The update system described here can be used with the XML subtree database system described in Lindblad I-A. The overall class object for a database system is referenced here as "xqe".

**[0055]** The update methods involve a two-stage mechanism. The first stage specifies some form of update, and the second step commits the update to the database. The update specifications are described as "*update values*". The "commit" function takes any sequence of *update values* and performs the specified changes to the database in a transactional manner. That means that either all the specified changes occur or none of them occurs. Some of the examples below comprise more than one query. Semicolons separate the queries and the system implicitly closes each query containing some update value with a "commit". Thus, each query transactionally completes prior to the next query where there are multiple queries.

Method: save

**[0056]** Save serializes an element node as an XML text file. For example, to serialize the value of the variable $node to the file "example.xml", the following method might be invoked:

```
xqe:save("example.xml", $node)
```

Method: load

**[0057]** Load returns an update *value* that, when committed, inserts a new document from an XML file. Optionally, a URI parameter labeling the loaded document can be

provided. For example, the following loads the (serialized XML) file "example.xml" to the database:

xqe:load("example.xml")

Method: document-insert

**[0058]** Document-insert returns an update *value* that, when committed, inserts a new document. For example, the following inserts a document:

xqe:document-insert("example.xml", <a>aaa</a>)

Method: document-delete

**[0059]** Document-delete returns an update *value* that, when committed, deletes a document from the database. For example, the following deletes a document:

xqe:document-delete("example.xml")

Method: node-replace

**[0060]** Node-replace returns an *update value* that, when committed, replaces a node. In a specific implementation, some constraints are applied to keep the data clean, such as rules that:

- attribute nodes cannot be replaced by non-attribute nodes,
- non-attribute nodes cannot be replaced by attribute nodes,
- element nodes cannot have document node children,
- document nodes cannot have multiple element node children, and
- document nodes cannot have text node children.

**[0061]** An example of a node-replace operation is:

```
xqe:document-insert("example.xml", <a><b>bbb</b></a>);
xqe:node-replace(document("example.xml")/a/b, <c>ccc</c>);
document("example.xml");
=>
<a><c>ccc</c></a>
```

**[0062]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a>bbb</a> into the database. The second query specifies an update which replaces the child "b" of the root element "a" of the

document with URI "example.xml" by a new node <c>ccc</c>. The third query returns the document node for the URI "example.xml" with the value shown.

Method: node-delete

**[0063]** Node-delete returns an *update value* that, when committed, deletes a node from the database. In a specific implementation, on-the-fly constructed nodes are not deletable. An example of a node-delete operation is:

```
xqe:document-insert("example.xml", <a><b>bbb</b></a>);
xqe:node-delete(document("example.xml")/a/b);
document("example.xml")
=>
<a/>
```

**[0064]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a>bbb</a> into the database. The second query specifies an update which removes the child "b" of the root element "a" of the document "example.xml". The third query returns the document node for the URI "example.xml" with the value shown.

Method: node-insert-before

**[0065]** Node-insert-before returns an *update value* that, when committed, adds an immediately preceding sibling to a node. In a specific implementation, some constraints are applied to keep the data clean, such as rules that:

- attribute nodes cannot be preceded by non-attribute nodes,
- non-attribute nodes cannot be preceded by attribute nodes,
- element nodes cannot have document node children,
- document nodes cannot have multiple element node children,
- document nodes cannot have text node children, and
- on-the-fly constructed nodes cannot be updated.

**[0066]** The arguments preferably specify individual nodes and not node sets. An example of a node-insert-before operation is:

```
xqe:document-insert("example.xml", <a><b>bbb</b></a>);
xqe:node-insert-before(document("example.xml")/a/b, <c>ccc</c>);
document("example.xml")
=>
<a><c>ccc</c><b>bbb</b></a>
```

**[0067]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a>bbb</a> into the database. The second query specifies an update which inserts before the child "b" of the root element "a" of the document "example.xml" a new node <c>ccc</c>. The third query returns the document node for the URI "example.xml" with the value shown.

Method: node-insert-after

**[0068]** Node-insert-after returns an update that, when committed, adds an immediately following sibling to a node. In a specific implementation, some constraints are applied to keep the data clean, such as rules that:

- attribute nodes cannot be preceded by non-attribute nodes,
- non-attribute nodes cannot be preceded by attribute nodes,
- element nodes cannot have document node children,
- document nodes cannot have multiple element node children,
- document nodes cannot have text node children, and
- on-the-fly constructed nodes cannot be updated.

**[0069]** The arguments preferably specify individual nodes and not node sets. An example of a node-insert-after operation is:

```
xqe:document-insert("example.xml", <a><b>bbb</b></a>);
xqe:node-insert-after(document("example.xml")/a/b, <c>ccc</c>);
document("example.xml")
=>
<a><b>bbb</b><c>ccc</c></a>
```

**[0070]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a>bbb</a> into the database. The second query specifies an update which inserts after the child "b" of the root element "a" of the document "example.xml" a new node <c>ccc</c>. The third query returns the document node for the URI "example.xml" with the value shown.

Method: node-insert-child

**[0071]** Node-insert-child returns an update item that, when committed, adds a new last child to a node. In a specific implementation, some constraints are applied to keep the data clean, such as rules that:

- only element nodes and document nodes can have children,
- element nodes cannot have document node children,
- document nodes cannot have multiple element node children,

- document nodes cannot have text node children, and
- on-the-fly constructed nodes cannot be updated.

**[0072]** The arguments preferably specify individual nodes and not node sets. An example of a Node-insert-child operation is:

```
xqe:document-insert("example.xml", <a/>);
xqe:node-insert-child(document("example.xml")/a,<b>bbb</b>);
document("example.xml")
=>
<a><b>bbb</b></a>
```

**[0073]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a></a> (with no content) into the database. The second query specifies an update which inserts a new child node bbb below the root element "a" of the document "example.xml". The third query returns the document node for the URI "example.xml" with the value shown.

**[0074]** As another example:

```
xqe:document-insert("example.xml", <a/>);
xqe:node-insert-child(document("example.xml")/a, attribute b { "bbb" });
document("example.xml")
=>
<a b="bbb"/>
```

**[0075]** In this example, the first query inserts a document with the URI "example.xml" having a root element <a></a> (with no content) into the database. The second query specifies an update which inserts a new attribute child node b="bbb" below the root element "a" of the document "example.xml". The third query returns the document node for the URI "example.xml" with the value shown.

Method: Commit

**[0076]** The commit function commits the update items included in the input (that is, actually makes the specified changes to the database), and returns the non-update items included in the input. If the result of every update query is automatically filtered by an implicit call to xqe:commit(), it need not be called explicitly to commit updates. However, xqe:commit() can be called explicitly to allow update queries to catch and handle errors that may occur. For example, a SOAP implementation might catch errors and report them back to the client using the SOAP error reporting mechanism.

[0077] After xqe:commit() has been called, any deadlock detected by the update query further accessing the database does not result in the update query being retried, but instead results in an error. Deadlock conditions can occur on any database access during the evaluation of an update query. Normally, update queries are automatically retried when a deadlock occurs. After xqe:commit() has been called, it would be incorrect to automatically retry the update query, so an error is signaled instead. Because of this, it is good practice for an update query to call commit() at most once. For example:

```
try {
 xqe:document-insert("example.xml", <a>aaa</a>)
} catch ($errInfo) {
 $errInfo
}
```

[0078] Fig. 13 describes the situation where an update encounters a link node. The dark nodes indicate link nodes. The node numbering describes the "document ordering of the nodes". The node numbered 1 corresponds to the root node of a subtree which contains the target of some *update value*. The function *copy-and-update* traverses this subtree applying update values from the change vector *V* as it encounters additional update target nodes. The traversal may reach a secondary link node as indicated at node 5. The target of link node 5 is another subtree which may contain additional nodes which need updating.

[0079] Each subtree includes in its data attributes a count of the number of nodes within the subtree. This count may be used to determine whether a given node id lies inside the subtree. This determination can be made by calculating a set of numeric inequalities between a given node id (the target of an update value) and the node id of the root of the subtree and the total number of nodes in the subtree.

[0080] In addition, when loading documents with a given subtree granularity, the XQE system inserts a certain amount of slack between the last subtree id used in a child subtree and the next subtree id used in the parent subtree. In this way, the system can insert additional nodes and subtrees into the child subtree without necessarily triggering any renumbering of the node ids in the parent subtree following the child subtree root in document order. In Fig. 13, the child subtree rooted at 5 is indicated to have (k) nodes, and the first following node in the parent subtree has a node id equal to $n > 5+k$.

[0081] If an insertion in a linked subtree overflows the preallocated node *id* slack, then all the following nodes in the parent subtree are copied with incremented node *ids*.

[0082] Fig. 14 describes the method for updating the subtree node counts and the subtree id slack. In each portion of Fig. 14, the top row of numeric variables describe the starting Subtree ids for one subtree (with slack) and the bottom row of numeric variables describes the actual Subtree node counts.

[0083] After xqe:commit() has been called, any deadlock detected by the update query further accessing the database does not result in the update query being retried, but instead results in an error. Deadlock conditions can occur on any database access during the evaluation of an update query. Normally, update queries are automatically retried when a deadlock occurs. After xqe:commit() has been called, it would be incorrect to automatically retry the update query, so an error is signaled instead. Because of this, it is good practice for an update query to call commit() at most once.

[0084] An example of a Commit operation is:

```
try {
   xqe:document-insert("example.xml", <a>aaa</a>)
} catch ($errInfo) {
   $errInfo
}
```

Stands and Forests

[0085] An XQE subtree-structured XML database system might aggregate subtrees into "stands" that are in turn aggregated in to "forests". Subtrees are inserted into a forest according to the following process. SubTree insertion occurs either when the data loader detects the end of a document or it completes a traversal of a configured subtree element. The first step in a process of adding a subtree to a forest is finding a suitable stand where the subtree can be added. At any given time, the forest manages a multiplicity of stands, some of which are on-disk stands (read-only objects backed by persistent disk file images that cannot be directly modified) and others are in-memory stands in the process being saved to disk as on-disk stands. In addition, there may be an in-memory stand available for update. If not, then a new in-memory stand is created for the purpose. The following steps then occur:

(1) compute index data
(2) compute *Classification* data
(3) lock *Stand* for update
(4) if the database is shutting down, then exit
(5) update the updateable *Stand*

(5.1) a *Journal* record is created

(5.2) serialize the *Subtree* data into the journal record

(5.3) put() *Subtree* data to *InMemoryStand*

(5.4) write *Journal* record

(5.5) set timestamps

(6) catch *Full* exception and either:

(6.1) flush *Stand* to *OnDiskStand*, and return to step (4), or

(6.2) exit if *Subtree* exceeds the maximum size of one *Stand*

(7) unlock *Stand*

Example of Update Algorithm

**[0086]** An example of an update algorithm will now be described with reference to Figs. 15-19. Fig. 15 illustrates a subtree that represents a new document fragment. Suppose an update operation is to insert the new document fragment into the structures shown in Figs. 7-10, specifically as child node under node 62, which is a "<c>" element, being a sibling between the <d> element and the <e> element. In an XML document, this would correspond to inserting the document fragment into the XML document of Fig. 7 at the point indicated by arrow 44 in Fig. 7. Locally, the new structure would be as illustrated in Fig. 16.

**[0087]** If the decomposition rules were that the decomposition occurs at "<c>" elements, then the subtree with a subtree label of "40" would undergo a refragmentation with node <c> 160 being the root node of the new subtree. Actually, two new subtree fragments would be formed, one representing portions of the added a document fragment (subtree label 60) and another or representing the remaining portions of the subtree labeled 40, which would now be referred to as subtree 70. New subtree fragment identifiers are allocated sequentially in the order that new subtrees are detected and written out to disk. When using facilities described in Lindblad I-A, the two new subtrees are output to a new stand and the old subtree 40 is marked deleted. A background process might merge stands by removing deleted subtrees, concatenating subtree fragment sets, and merging parent-child index lists.

**[0088]** Preferably, nodes in the subtree-structured-XML database are recoverable in "document order", to facilitate XQuery processing. To do this, each node is assigned a numeric ordinal value, such as a 64-bit number. These ordinals maintain the invariant property that one node comes before another node in the document if and only if the ordinal of the one node is less than the ordinal of the other node. Herein "document order" is defined as the order in which nodes appear when viewed as an actual XML document, from top to

bottom. For example, the tree 180 and XML document 182 shown in Fig. 18A represent a document where the document order of nodes is <a>, <b>, <c>, <d>, <e>.

**[0089]** Ordinal values can be assigned a number of different ways. But one example allocates ordinals using higher order bits at the outset and lower order bits as needed for updates. This is illustrated in Fig. 18B for the tree and document shown in Fig. 18A. As illustrated there, the nodes are allocated as a sequential multiples of 2^32. The lower order 32 bits can then be used to interpolated ordinal's between existing ordinal sequences in order to maintain the invariant property described above across insertions and deletions of nodes, as illustrated in Fig. 19.

**[0090]** Fig. 19A illustrates portions of the structure used as an example above. Note that each of the nodes is allocated a sequential multiple of 2^32. When nodes are added, the new nodes can be assigned ordinals that fall between the ordinals that would be before and after the inserted fragment, with the new nodes having a value that it is a multiple of 2^32 plus a multiple of 2^16, thereby allowing for later inserts. Of course, at some point after long the sequences of repeated inserts, the system could run out of ordinals and future inserts that require unavailable ordinals would either be prohibited or the ordinals could be all reassigned. One approach to ordinals reassignment is to run through the database with the wave of changes. Another approach is to write out the entire database as one or more XML documents, and then read the one or more XML documents back in, to populate the subtree-structured XML database anew.

**[0091]** Embodiments of the present invention provide an XML database with updatability. When XML data is modified, only a small number of subtrees typically need to be revised. Data compression can also be provided, e.g., by using atoms to represent text data, as well as by applying additional compression techniques when data is written to disk and decompression techniques when data from disk is read into memory to be processed. Queries may be processed efficiently by applying the query to groups of subtrees (i.e., stands) and aggregating the results.

**[0092]** While the invention has been described with respect to specific embodiments, one skilled in the art will recognize that numerous modifications are possible. The data structures described herein can be modified or varied; particular contents and coding schemes described herein are illustrative and not limiting of the invention. Any or all of the data structures described herein (e.g., forests, stands, subtrees, atoms) can be implemented as objects using CORBA or object-oriented programming. Such objects might contain both data structures and methods for interacting with the data. Different object classes (or data

structures) may be provided for in-scratch, in-memory, and/or on-disk objects. Examples of methods are described and some objects might have more or fewer objects.

**[0093]** Additional features to support portability across different machines or different file system implementation, random access to large files, concurrent access to a file by multiple processes or threads, various techniques for encoding/decoding of data, and the like can also be implemented. Persons of ordinary skill in the art with access to the teachings of the present invention will recognize various ways of implementing such options.

**[0094]** Various features of the present invention may be implemented in software running on general-purpose processors, dedicated special-purpose hardware components, and/or any combination thereof. Computer programs incorporating features of the present invention may be encoded on various computer readable media for storage and/or transmission; suitable media include suitable media include magnetic disk or tape, optical storage media such as compact disk (CD) or DVD (digital versatile disk), flash memory, and carrier signals adapted for transmission via wired, optical, and/or wireless networks including the Internet. Computer readable media encoded with the program code may be packaged with a device or provided separately from other devices (e.g., via Internet download).

**[0095]** Thus, although the invention has been described with respect to specific embodiments, it will be appreciated that the invention is intended to cover all modifications and equivalents within the scope of the following claims.

## Appendix A. Subroutines For Node Updating

```
for each node update u(n) in change vector V:
    find the subtree S containing the node n;
    copy-and-update(V, S, u.root());

// recursive subtree traversal
copy-and-update(ChangeVector V, Subtree S, Node node):

    if (exists node-replace for node) then replace node; return;

    for (each node-insert-before(node, n))
      insert new node n into output subtree;
      remove node-insert-before update value from change vector V;
      break;

    switch (nodeKind(node)):
      case ElemNodeKind:
        for (attr in node attributes) copy-and-update(V, S, attr);

        for (child in node children)  copy-and-update(V, S, child);

        for (each node-insert-child(node))
          insert new child to node in output subtree;
          remove node-insert-child update value from change vector V;
        break;

      case DocNodeKind:
        for (child in node.children()) copy-and-update(V, S, child);
        break;

      case LinkNodeKind:
        if (link to parent = first node in tree) then
          copy node to output tree;
        else
          check for updates in link target subtree;
          if (no updates in link target subtree) then
            copy link node to output subtree;
          else
            copy-and-update(V, link target subtree, link target subtree root) ;

          for (each node-insert-after(link target, n))
            insert n after parent in output subtree;
            remove node-insert-after update value from change vector V;
        break;

      case TextNodeKind:
      case PINodeKind:
      case CommentNodeKind:
```

```
        copy node to output subtree;
        break;
    end-switch

    if (parent(node) is not a link node) then
      for (each node-insert-after(node, n))
        insert n after node in output subtree;
        remove node-insert-after update value from change vector V;

end-copy-and-update
```

WHAT IS CLAIMED IS:

1. In an XML handling system, wherein XML documents are stored in structured forms, a method of updating an XML document without requiring global changes to the XML document, the method comprising:

organizing a representation of the XML document as a collection of subtrees, wherein a subtree represents a connected set of one or more nodes and wherein a node represents an XML element, content, attribute or value;

identifying an affected set comprising subtrees that would be affected by an update instruction;

creating a replacement set of one or more subtrees that would substitute for the subtrees in the affected set;

adding the replacement set to the representation; and

marking each of the subtrees in the affected set as being no longer part of the representation.

2. The method of claim 1, wherein marking a subtree as being no longer part of the representation comprises setting a delete flag for the subtree.

3. The method of claim 1, further comprising assigning an ordinal value to each node such that if and only if a first node comes before a second node in the XML document, the ordinal value assigned to the first node is less than the ordinal value assigned to the second node.

4. The method of claim 3, wherein the ordinal values are assigned as multiples of a number greater than one such that unassigned ordinal values exist between each initially assigned ordinal value, thereby providing for ordinal values that could be assigned to subsequently inserted nodes.

5. The method of claim 1, wherein the update instruction is one of Save, Load, Document-insert, Document-delete, Node-replace, Node-delete, Node-insert-before, Node-insert-after, Node-insert-child and Commit.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 December 2003 (24.12.2003)

PCT

(10) International Publication Number
WO 2003/107576 A3

(51) International Patent Classification[7]: G06F 17/30, 7/00, 17/00, 17/21, 17/24, 15/00

(21) International Application Number: PCT/US2003/018788

(22) International Filing Date: 13 June 2003 (13.06.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/389,052 13 June 2002 (13.06.2002) US

(71) Applicant: CERISENT CORPORATION [US/US]; 2000 Alameda de las Pulgas, Suite 100, San Mateo, CA 94403-1269 (US).

(72) Inventors: LINDBLAD, Christopher; 35 Live Oak Road, Berkeley, CA 94705 (US). PEDERSEN, Paul; 1788 Oak Creek #310, Palo Alto, CA 94304 (US).

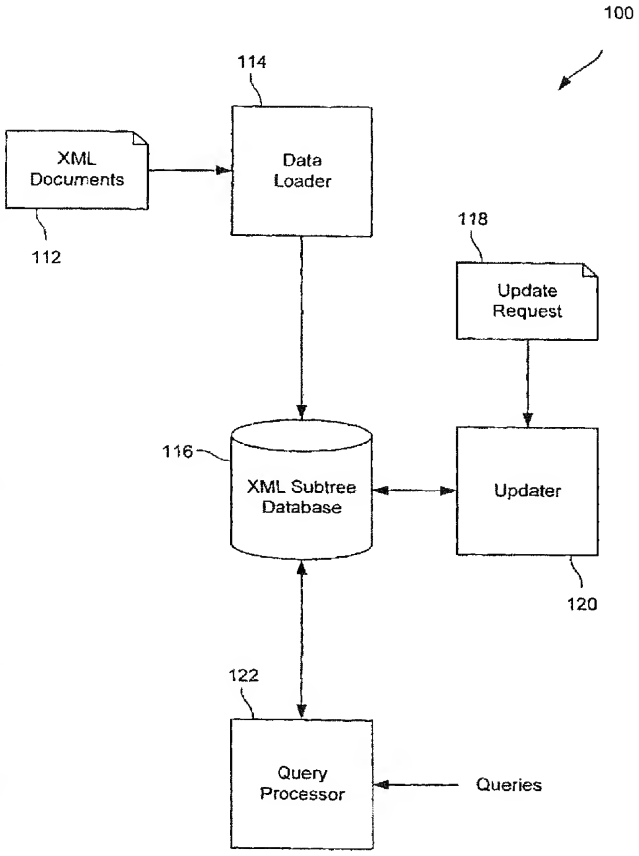
(74) Agents: ALBERT, Philip, H. et al.; Townsend and Townsend and Crew LLP, Two Embarcadero Center, 8th Floor, San Francisco, CA 94111 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT (utility model), AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ (utility model), CZ, DE (utility model), DE, DK (utility model), DK, DM, DZ, EC, EE (utility model), EE, ES, FI (utility model), FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK (utility model), SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO,

*[Continued on next page]*

(54) Title: XML-DB TRANSACTIONAL UPDATE SYSTEM




(57) Abstract: In an XML handling system, point updates to an element of an XML document stored in the database is possible (11). Updates include addition or deletion of whole documents, addition of a child node to any element node (this includes attribute nodes), the addition of new siblings to any element node, the deletion of any element node, and the replacement of any node by a new node (120). The database system might include a set of functions that can be invoked to affect an update (i.e., an addition, deletion or modification). Such updates can be submitted as queries, such as instructions within an XQuery query (122).

WO 2003/107576 A3

SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

**(88) Date of publication of the international search report:**
8 April 2004

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/US03/18788

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 17/30, 7/00, 17/00, 17/21, 17/24, 15/00
US CL : 707/1,10,100; 715/513
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/1,10,100, 203; 715/513,523

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X<br>---<br>Y | US 2002/0023113A (Hsing et al.) 21 February 2002 (21.02.2002), column 4, lines 3-24, column 8 to column 9, line 23 | 1, 2, 5<br>---<br>3, 4 |
| Y,P | US 2003/0110150A1 (O'Neil et al) 12 June 2003 (12.06.2003), column 17 , lines 13-16 | 3, 4 |

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 22 January 2004 (22.01.2004) | 29 JAN 2004 |
| Name and mailing address of the ISA/US<br>Mail Stop PCT, Attn: ISA/US<br>Commissioner for Patents<br>P.O. Box 1450<br>Alexandria, Virginia 22313-1450<br>Facsimile No. | Authorized officer<br>Safet Metjahic for Michelle L. Owen<br>Telephone No. 703-308-1436 |

Form PCT/ISA/210 (second sheet) (July 1998)

# INTERNATIONAL SEARCH REPORT

PCT/US03/18788

**Continuation of B. FIELDS SEARCHED Item 3:**
EAST
Search terms: XML, tree, hierarchical, update, flag, ordinal value, identifier, node, delete, insert, replace

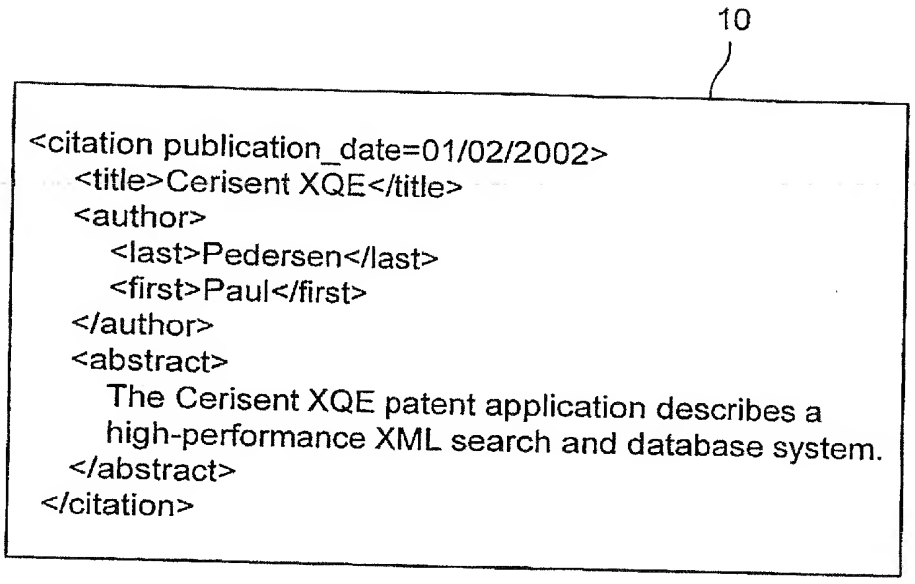



FIG. 1 (Prior Art)

```
[01]  NAMESPACE  name_1  =  "uri-string_1"
[02]  NAMESPACE  name_2  =  "uri-string_2"
[03]  ...
[04]  DEFAULT ELEMENT NAMESPACE = "default-element-uri-string"
[05]  DEFAULT ELEMENT NAMESPACE = "default-function-uri-string"
[06]
[07]  DEFINE FUNCTION function_a(datatype $arg_a1, Datatype $arg_a2,...)
[08]  RETURNS  { function_expression_a }
[09]
[10]  DEFINE FUNCTION function_b(datatype $arg_b1, Datatype $arg_b2,...)
[11]  RETURNS  { function_expression_b }
[12]  ...
[13]  FOR $variable_a1 IN expression_a2, variable_a3, IN expression_a4,...
[14]  LET $variable_b1 := expression_b2, variable_b3, := expression_b4,...
[15]  FOR $variable_c1 IN expression_c2, variable_c3, IN expression_c4,...
[16]  LET $variable_d1 := expression_d2, variable_d3, := expression_d4,...
[17]  ...
[18]  WHERE  where_expression
[19]  RETURN  return_expression
[20]  SORTBY  sortby_expression
```

FIG. 2 (Prior Art)

SUBSTITUTE SHEET (RULE 26)

30

```
<citation>
   <title>Cerisent XQE</title>
   <author>
      <last>Pedersen</last>
      <first>Paul</first>
   </author>
   <abstract>  The document describes an XML
               search and query system
   </abstract>
</citation>
```

FIG. 3

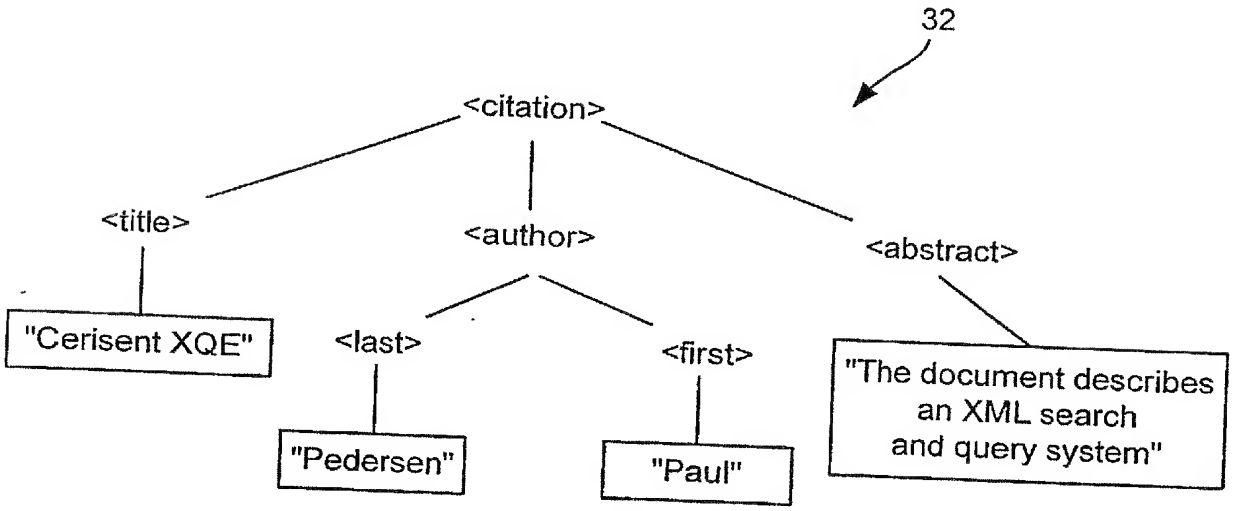



FIG. 4A

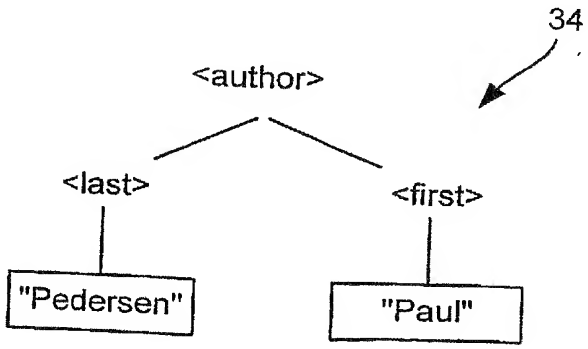



FIG. 4B

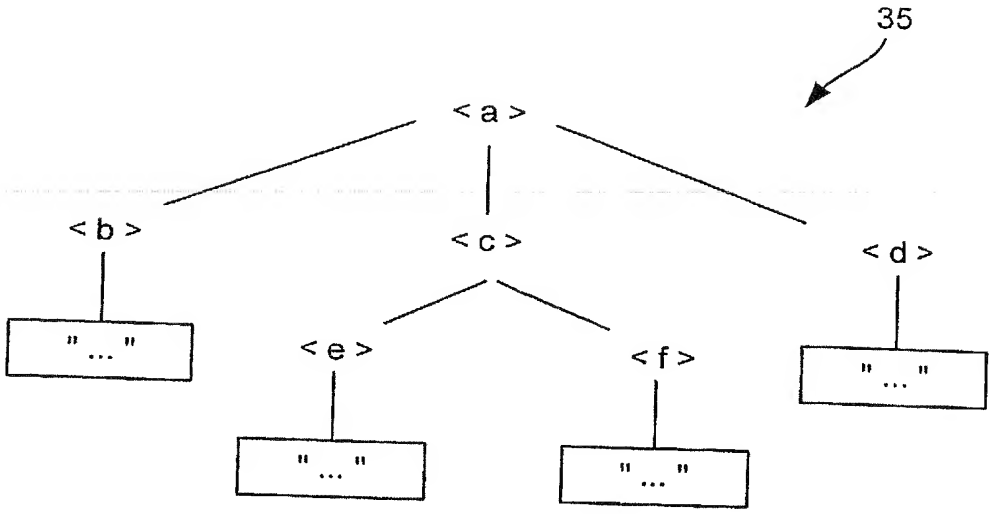



FIG. 5

< b K = "v" > node text</b>

FIG. 6A

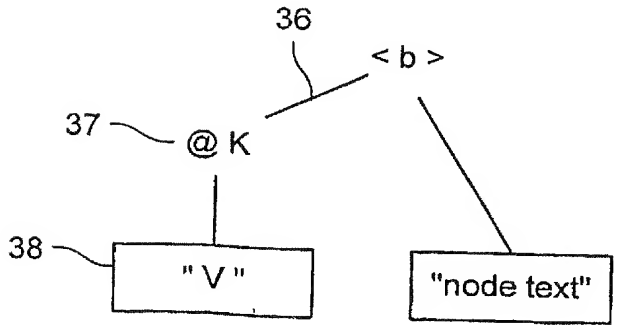



FIG. 6B

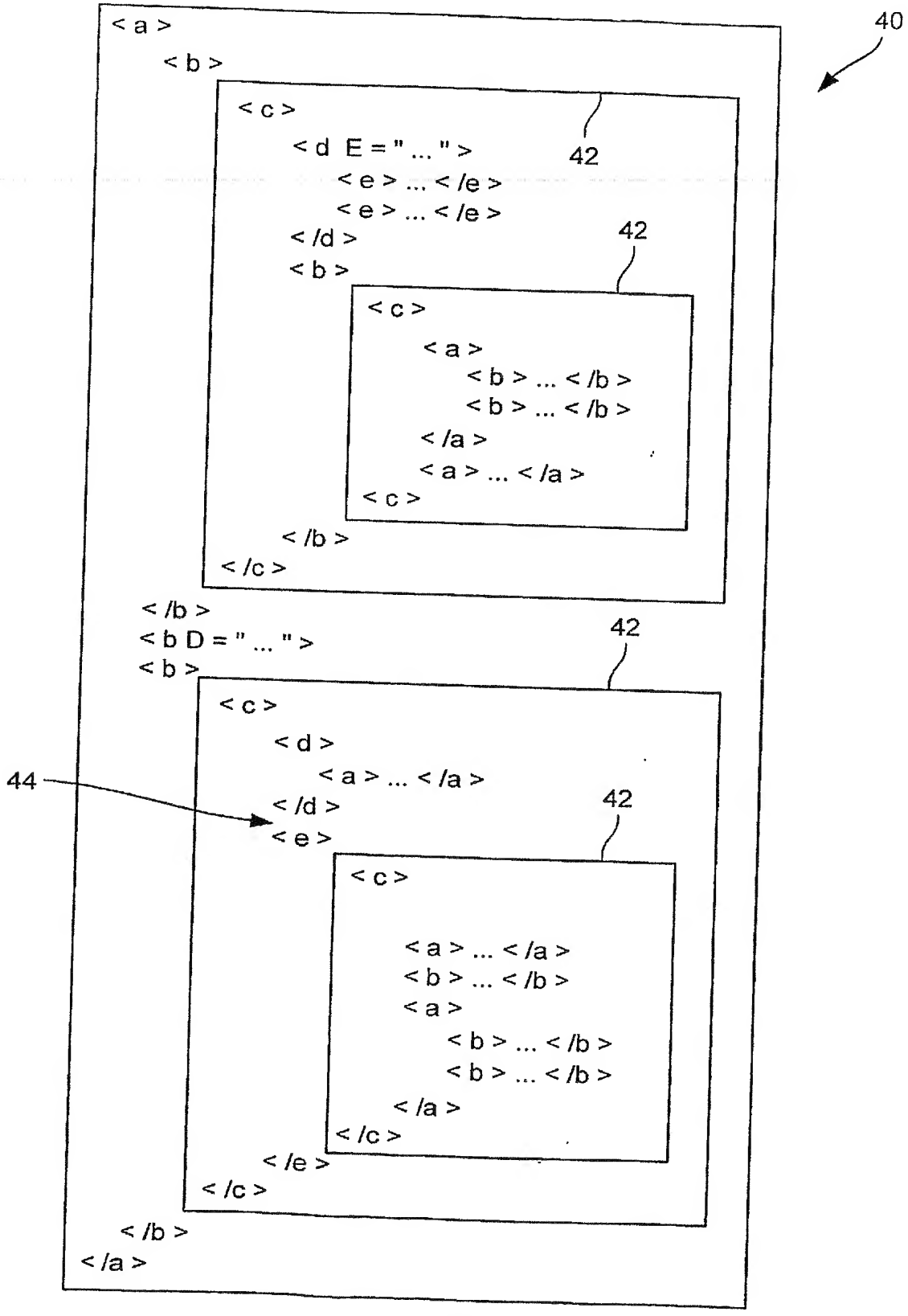



FIG. 7

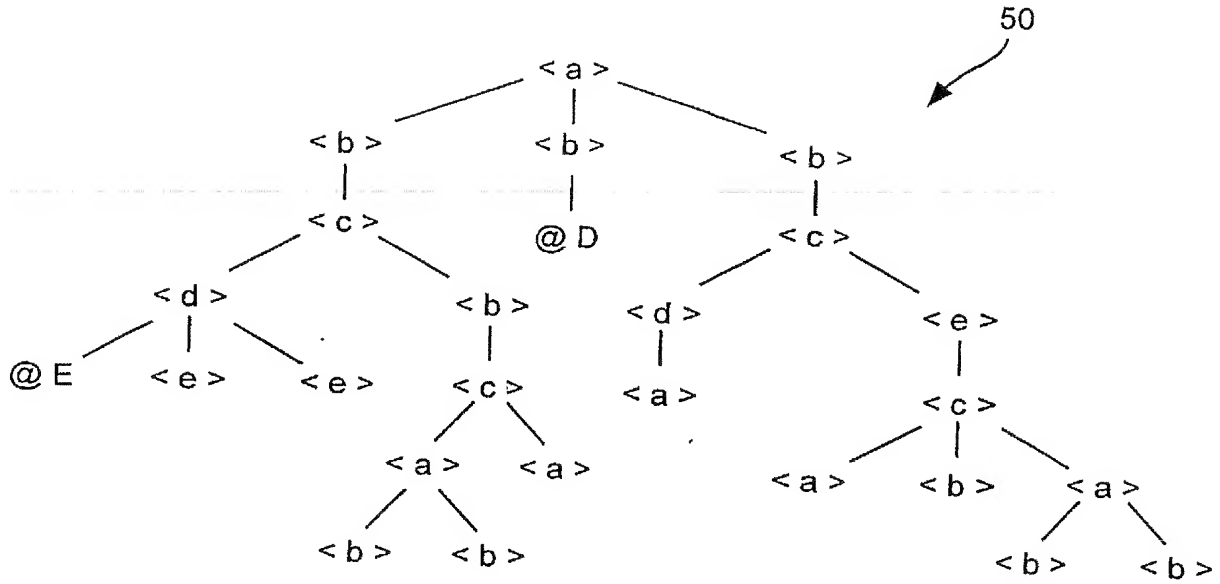



FIG. 8

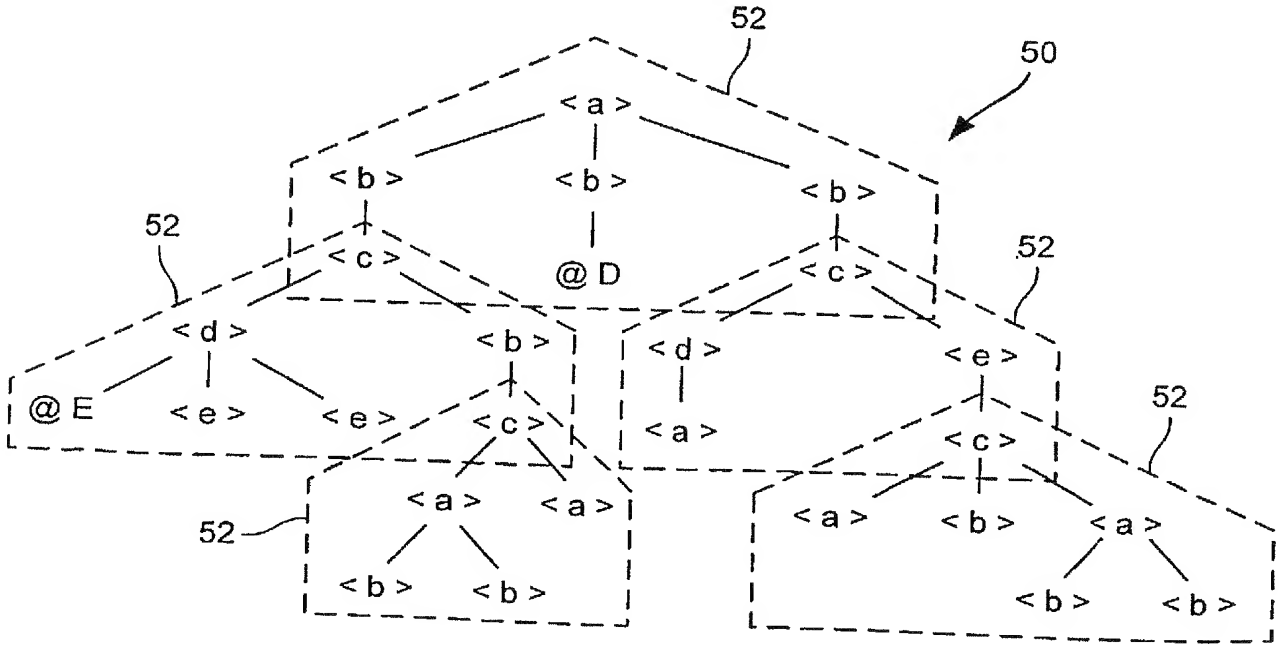



FIG. 9

SUBSTITUTE SHEET (RULE 26)

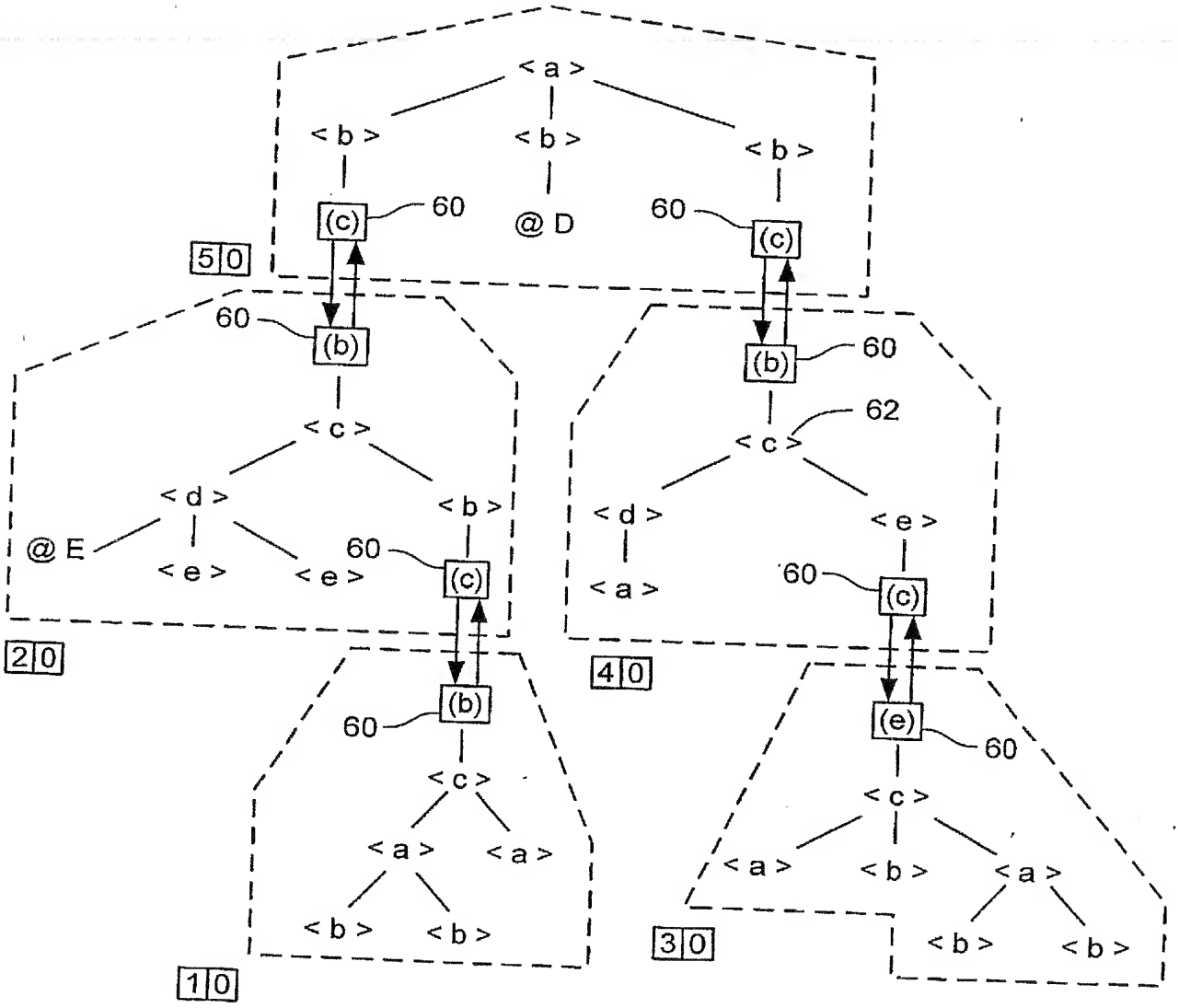



FIG. 10

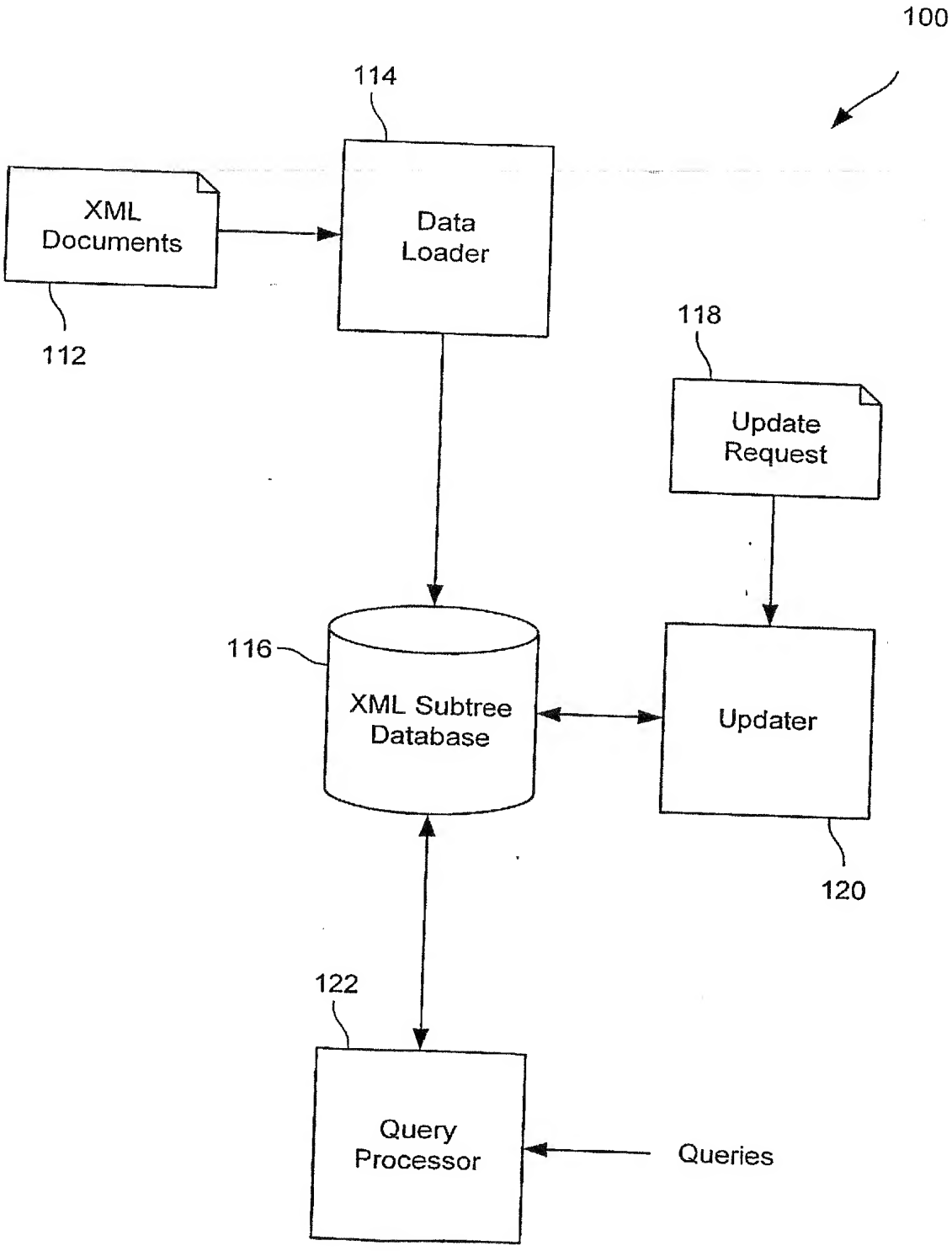



FIG. 11

SUBSTITUTE SHEET (RULE 26)

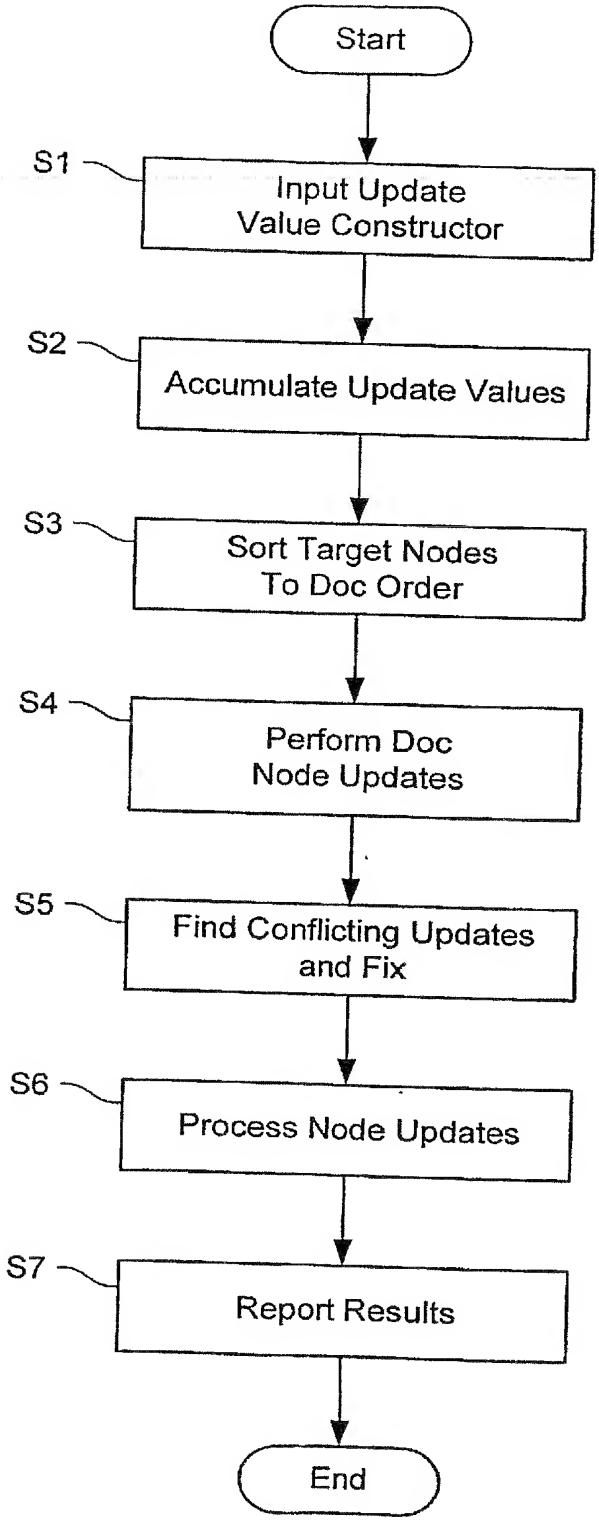



FIG. 12

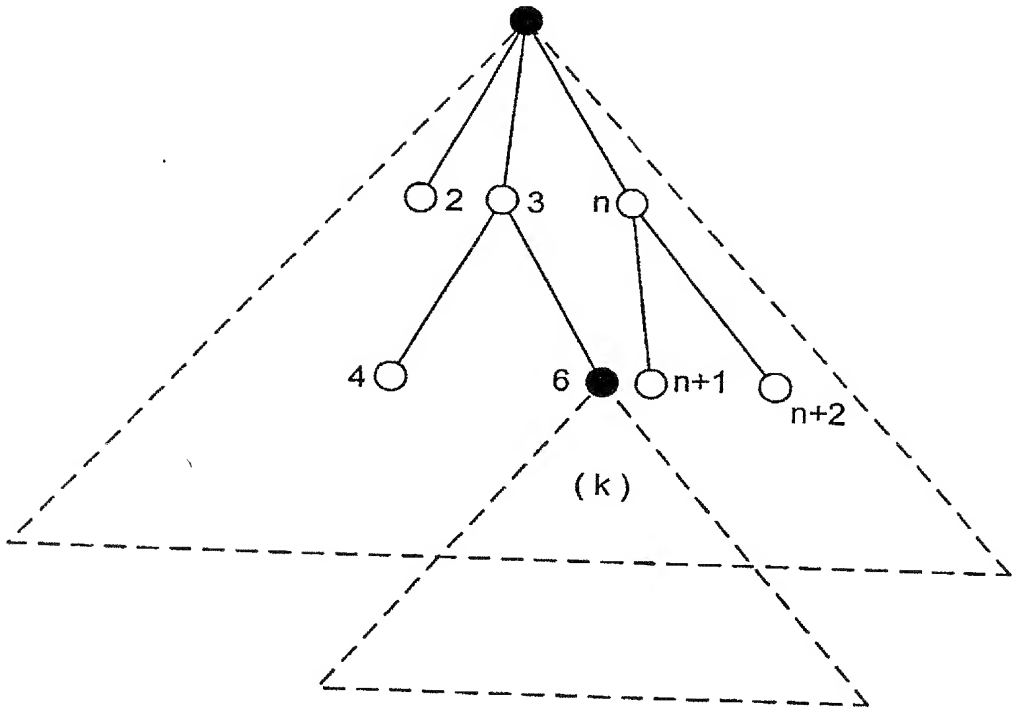
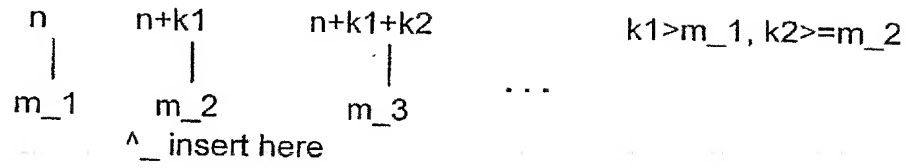


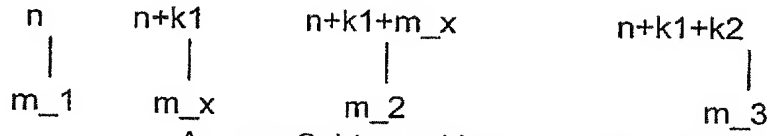

FIG. 13

Case Insert (preceeding or following):

state before:



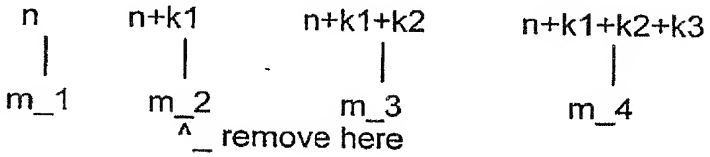

state after:



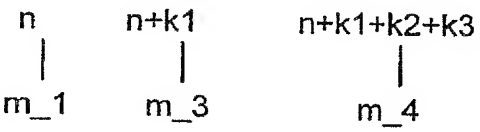

if (k1+k2<m_x) stop
else re-write subtree, recursively updating root original

Case Remove

state before:

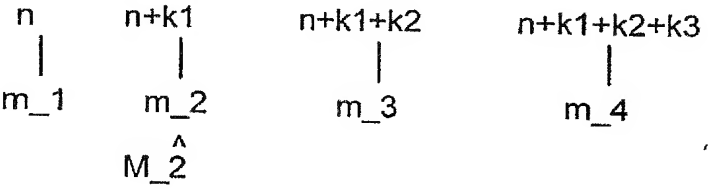



state after:

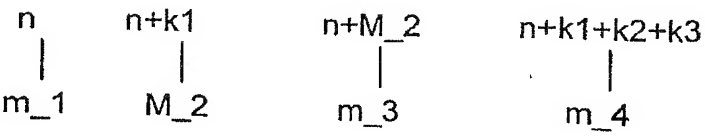



Case Replace

state before:




state after:




If (M_2>k2)// slack overflow
re-write m_3 node, and if necessary continue:

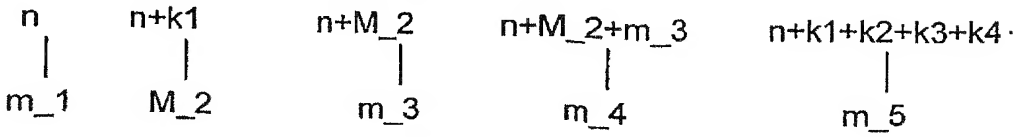



FIG. 14

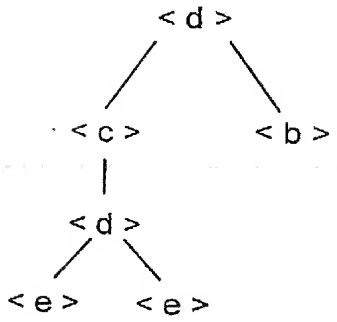



FIG. 15

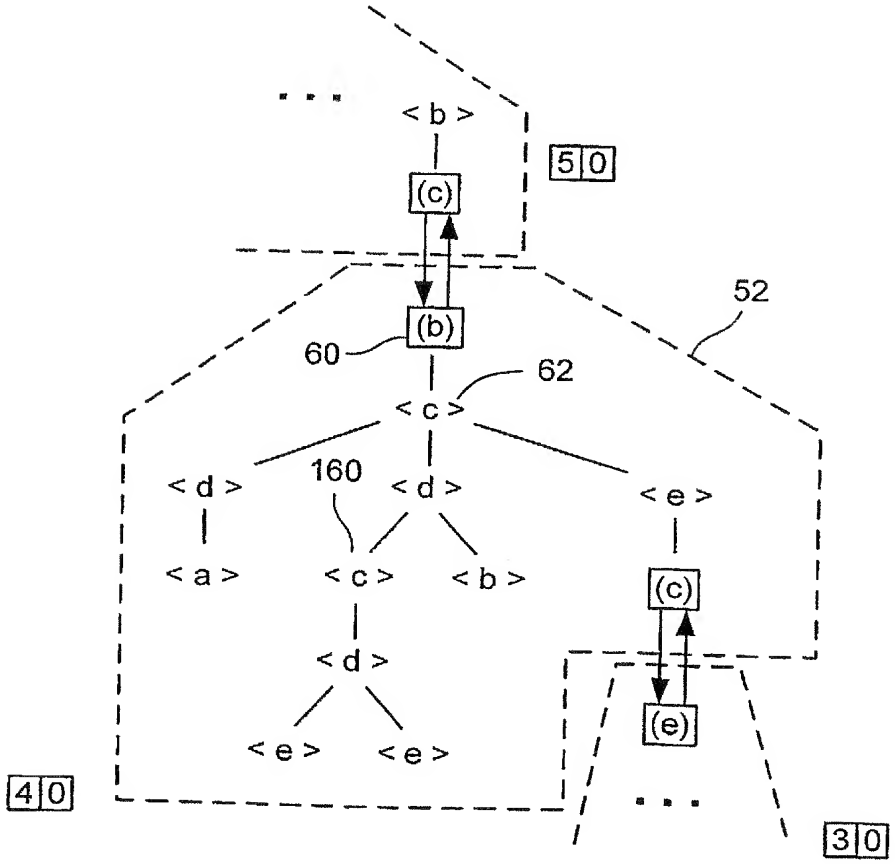



FIG. 16

SUBSTITUTE SHEET (RULE 26)

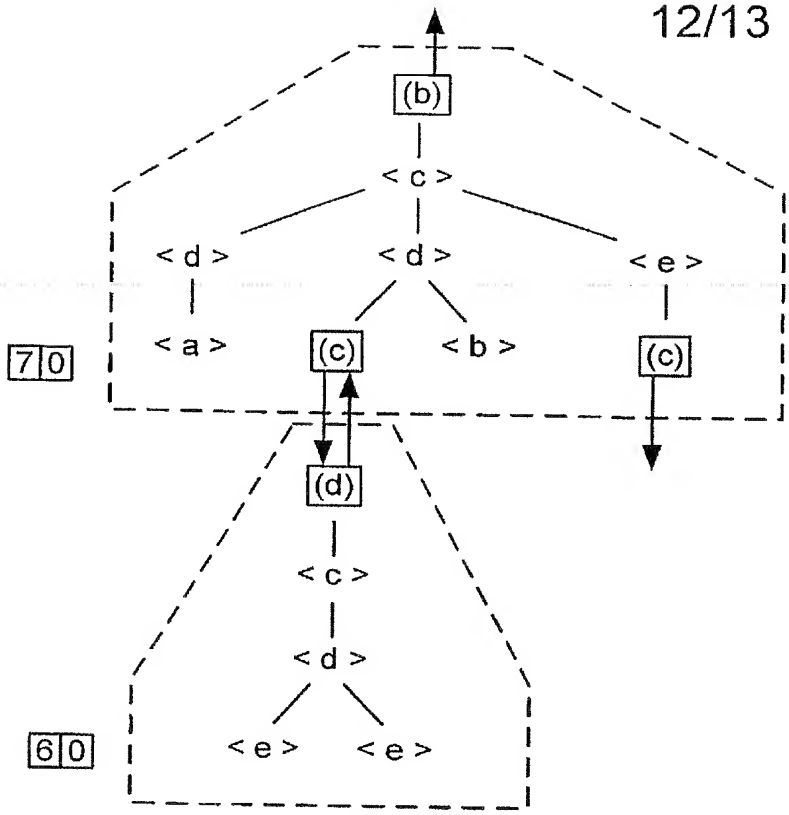



FIG. 17

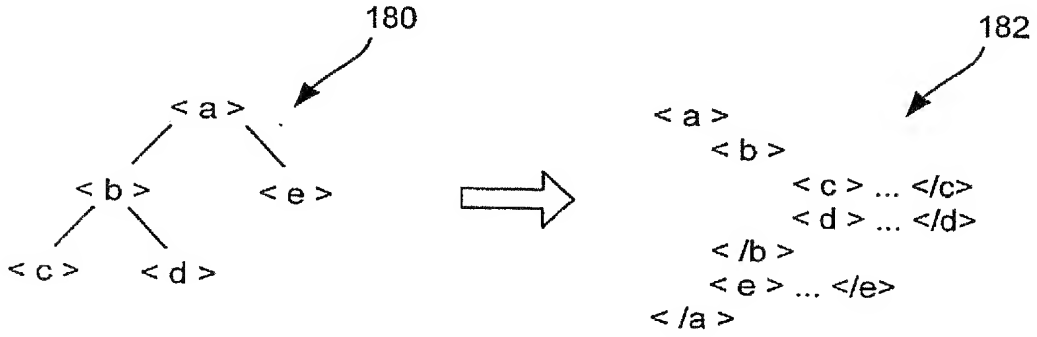



FIG. 18A

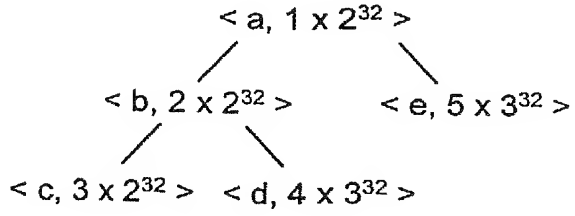



FIG. 18B

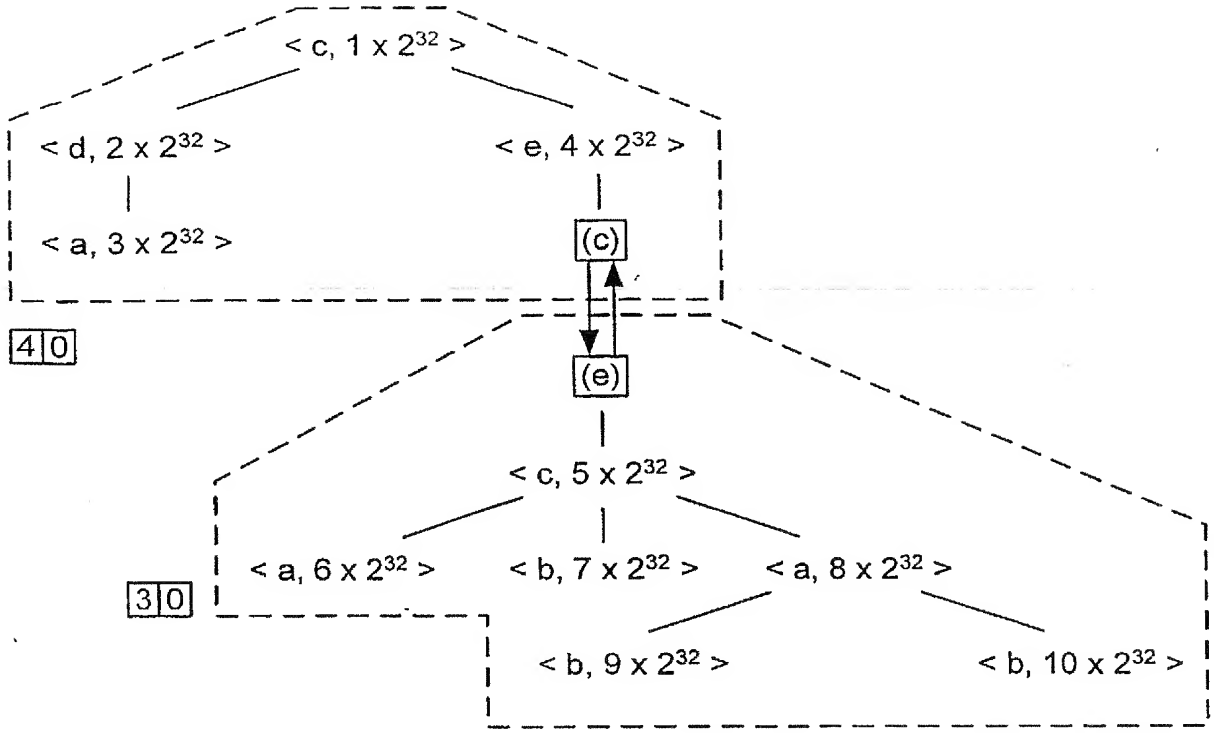



FIG. 19A

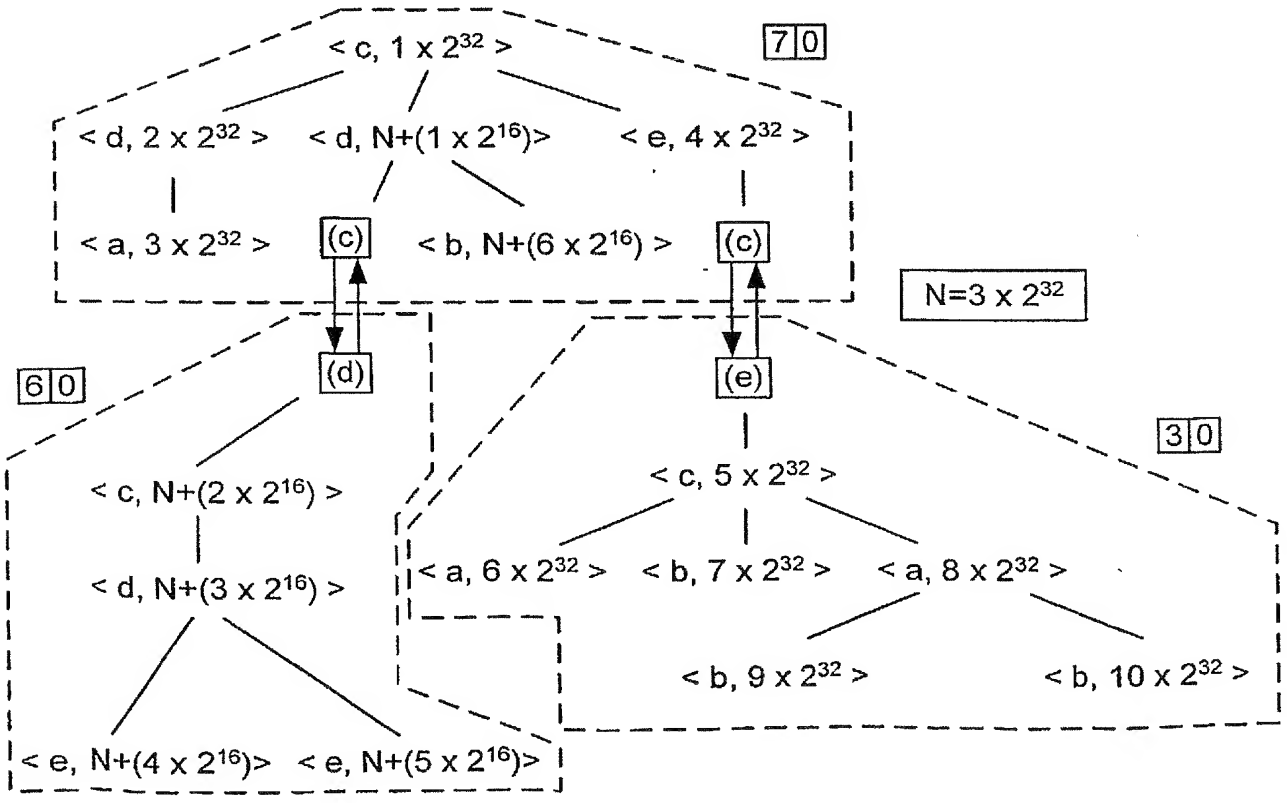



FIG. 19B